A Comparison of Some Dimension Reduction Techniques with Varied Parameters

By

Nicholas C. Oderio

Submitted to the graduate degree program in Department of Mathematics and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Arts.

Dr. Bozenna Pasik-Duncan, Chairperson

Committee members

Dr. Tyrone Duncan

Dr. Erik Van Vleck

Date defended: December 1, 2017

The Thesis Committee for Nicholas C. Oderio certifies
that this is the approved version of the following thesis :

A Comparison of Some Dimension Reduction Techniques with Varied Parameters

Dr. Bozenna Pasik-Duncan, Chairperson

Date approved: December 1, 2017

Abstract

This paper presents and explains several methods of dimensionality reduction of data sets, beginning with the well known PCA and moving onto techniques that deal with data on a nonlinear manifold. Methods for handling data whose underlying structure is a nonlinear manifold are separated by whether or not sparse matrices are involved in the computation. Additionally, the methods discussed are demonstrated and compared by running them on data sets whose underlying structure is known. Results from same methods with different values for input parameters are also examined. Finally, some results on a small set of Persyst EEG data collected as a part of the Epilepsy Bioinformatics Study for Antiepileptogenic Therapy from the Laboratory of Neuro Imaging at USC Stevens Institute of Neuroimaging and Informatics in the Keck School of Medicine of USC is analyzed using some of these methods.

Acknowledgements

I must express my sincerest gratitude to my advisor, Dr. Bozenna Pasik-Duncan. Her involvement, leadership, encouragement, professionalism, positivity, and support have been sources of inspiration and dedication to my studies. I am extremely fortunate to have been able to work with such an enthusiastic mentor.

I would like to thank Dr. Tyrone Duncan for his willingness to provide assistance when necessary and for his contributions to stochastic adaptive control, which made for some wonderful lectures my first semester of graduate school. I am also indebted to Dr. Erik Van Vleck, for introducing me to the wonderful subject of numerical analysis, always showing interest in whatever questions I brought to him, and general excitement.

A special thanks to Dr. Dominique Duncan and the Laboratory of Neuro Imaging at USC Stevens Institute of Neuroimaging and Informatics in the Keck School of Medicine of USC for hosting me during an illuminating trip to see how a large repository of EEG data is collected and analyzed at the institute.

A special thanks to Mike Stees as well for helping me with various Matlab questions.

Lastly, I would like to thank my friends and family, especially my parents, for their continued love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the modern world, demand for data analysis is growing, especially for large data sets. There has even been a rise in use of the term “big data." While the term is somewhat vague, large data sets can be large in both number of observations and number of variables measured. While a large number of samples is generally desirable, a large number of variables can make data analysis and visualization difficult. One strategy for handling such data sets is to reduce the dimension of the data set. In particular, reducing the dimension of the number of variables measured while retaining as much of the original information as possible.

Principal Component Analysis (PCA) is a method for accomplishing the task of dimension reduction. PCA is a well known technique and commonly used in real world applications to compress data sets so that they may be visually represented and interpreted. It is worth noting that PCA is a mostly descriptive, as opposed to inferential, technique [1].

In this paper, the method of PCA is discussed, including when it is appropriate to use and its shortcomings. Naturally, then, methods that are modifications of or based on PCA are introduced, and discussed similarly. These methods include kernel PCA, diffusion mapping, local linear embedding, and Laplacian eigenmaps. These methods were implemented in Matlab and run using test data.

1.1 Principal Component Analysis

The idea behind PCA is as follows: reduce the number of dimensions (i.e., measured variables) so that the original data are represented in fewer (ideally one, two, or three dimensions) while

preserving as much of the original structure of the data as possible in a linear subspace of the original data set. What this amounts to is expressing the given data points as linear combinations of the given predictor variables. These new expressions form the principal components and can be used to map the original data into a lower dimensional space.

Before we see how the principal components are formed, it is important to understand what structure is preserved. In the case of PCA, the goal is to retain as much of the original variance as possible; that is, find linear combinations of the original predictor variables with maximum variance. Thus, the variability of the original data is what PCA aims to capture. PCA also requires that each subsequent linear combination of the original variables be uncorrelated with every other linear combination. Essentially, every principal component is orthogonal to every other principal component, resulting in perpendicular axes in the lower-dimensional space .

Suppose we are given a data set X of size $n \times p$, where n is the number of of observations and p is the number of variables. Thus, each row of X is one set of observations, and each column of X is all recorded values of a particular variable.

It is likely that not every variable measured will have the same units. If this is the case, the data matrix X can be be transformed so that each entry becomes its standard normal equivalent. This is done by subtracting the column mean from each element in that column and then dividing each column by its standard deviation. In fact, it is recommended that, regardless, the column means be subtracted from each respective column. This allows singular value decomposition to be performed on the column-centered data matrix which is equivalent to PCA [1] .

Now, the covariance matrix must be formed. Here, the sample covariance is used, given by

$$C = \frac{1}{n-1} X^T X$$

Note that $X^T X$ yields a square matrix and is both real and symmetric.

Next, the eigenvalue problem

$$Cu = \lambda u$$

must be solved for eigenvalues λ and corresponding u eigenvectors u, subject to the constraint $u^T u = 1$. This constraint can also be thought of as letting each eigenvector be a column of a matrix U and for each column of U, the sum of the squares of each element is equal to one, i.e. $\|u_j\| = 1 \quad \forall j$ [1]. Furthermore, when solving for the eigenvalues and eigenvectors, the eigenvalues should be somehow stored so that the eigenvalues are in descending order. Consequently, the columns of U should be ordered so that the first column corresponds to the largest eigenvalue, etc.

Once the eigenvalues and corresponding eigenvectors of the (sample) covariance matrix C are found, the data can be transformed. Using the centered or standardized data matrix X, the new coordinates in the lower dimensional space are given by

$$Y = XU$$

The rows of the matrix Y give the new coordinates for each data vector. The matrix Y will have p columns, of which the first q are kept, though for the purpose of visualization q is typically two or three [1]. Note that the sum of all the eigenvalues from PCA gives the total variance of the original data set. Thus, the proportion of the total variance explained by each principal component can be calculated from the eigenvalues λ_i and is given by

$$\frac{\lambda_i}{\sum_{i=1}^{p} \lambda_i}$$

For example, if one wanted to know how much of the original variance was retained for say, two principal components, this percentage is found by computing

$$\frac{\lambda_1 + \lambda_2}{\sum_{i=1}^{p} \lambda_i}$$

where λ_1 and λ_2 are the two largest eigenvalues.

It was mentioned previously that PCA is equivalent to singular value decomposition of a column-centered matrix. This is important because some programming languages, Matlab in par-

ticular, have built-in functions that perform PCA by using singualr value decomposition. The equivalence is shown in [1] and summarized below.

Let $\hat{X}$ represent the column-centered data matrix. An arbitrary $n \times p$ matrix may be written as

$$\hat{X} = ALB^T$$

where A is $n \times r$, B is $p \times r$, and L is an $r \times r$ diagonal matrix, where r is the rank of $\hat{X}$. Additionally, let $A^T A = B^T B = I$. It is assumed that the diagonal elements of L are in decreasing order. The columns of B are the right singular vectors of $\hat{X}$ and give the vectors u_j. Using the property $B^T B = I$, the principal components of $\hat{X}$ are given by $\hat{X}B = ALB^T B = AL$. Put another way,

$$(n-1)C = \hat{X}^T \hat{X} = (ALB^T)^T (ALB^T) = BLA^T ALB^T = BL^2 B^T$$

where the elements on the diagonal of L^2 give the variance of each principal component.

The main advantages of PCA are not difficult to see. The entire purpose of dimension reduction is to condense data so that it is easier to interpret and visualize. PCA does this well, and in a certain sense one can always figure out how well by looking at the proportion of variance explained by the number principal components we choose to keep. Additionally, the distribution of the given data does not matter. Indeed, we started with an arbitrary data matrix and were able to perform the entire analysis. The major drawback of PCA is discussed in the beginning of the next section.

1.2 Nonlinear Full Matrix Methods

An underlying assumption of PCA is that the original data are somehow linearly related or already exist on a linear manifold. This may not always be the case. However, it may be the case that a given data set exists on a nonlinear manifold that is linear in some lower dimensional space (imagine a rolled up sheet of paper being unrolled). For data with such a nonlinear relation, there still exist techniques to reduce dimensionality. The following methods discussed all aim to accomplish

this task. The main distinction between them is that first two, kernel PCA and diffusion mapping, use full matrices. Local linear embedding (LLE) and Laplacian eigenmaps make use of a sparse matrix in the computation.

1.2.0.1 Kernel PCA

As previously mentioned, a shortcoming of traditional PCA is that it assumes the data are already on a linear manifold. A nonlinear manifold does not prevent PCA from being performed, but the method might not perform particularly well. Supposing that given data were sampled from a nonlinear manifold, it would be nice if PCA could be modified so that an analysis similar to before could be conducted. Kernel PCA (kPCA) is a method that adheres to this idea. With that motivation, kPCA maps the original data into a higher dimensional space using a kernel function to represent the inner products between data points. The inner products of the data points in the higher dimensional space are then stored in a kernel matrix and find the eigenvectors of this matrix instead of the original covariance matrix. This method was first proposed by [2] and is outlined here.

The first thing to do is to form the kernel matrix. In order to do this, a kernel function must be chosen. There are many such functions, although common choices are the linear kernel (which makes this method equivalent to traditional PCA), polynomial kernel, and Gaussian kernel. The polynomial kernel is given by

$$\kappa(x_i, x_j) = (x_i \cdot x_j^T + c)^p$$

and the Gaussian kernel is given by

$$\kappa(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$$

In the polynomial kernel, c is a constant and p is the degree of the polynomial.

After choosing a kernel, the kernel matrix K is formed and will be size $n \times n$. Note that for the kernel matrix K, this gives that $K_{ij} = \Phi(x_i) \cdot \Phi(x_j)^T$ [3] for a map Φ (see below). To find the

eigenvectors as before, the kernel matrix must be centered in the feature space. However, this must be done implicitly. Fortunately, this can be expressed, and so computed, in terms of K as shown in [4]. Indeed, suppose a map Φ is used to project the data into a high dimensional feature space $\mathscr{F}$, $\Phi : \mathbb{R}^p \to \mathscr{F}$ and $x_i \to \Phi(x_i)$. To center the data, let

$$\hat{\Phi}(x_i) := \Phi(x_i) - \frac{1}{n}\sum_{i=1}^{n}\Phi(x_i)$$

Then $\hat{K}_{ij} = \hat{\Phi}(x_i)\cdot\hat{\Phi}(x_j)$ and the problem to be solved is $\hat{K}v = \lambda v$. Substituting in the definition of $\hat{\Phi}$, we get

$$\hat{K}_{ij} = (\Phi(x_i) - \frac{1}{n}\sum_{k=1}^{n}\Phi(x_k))\cdot(\Phi(x_j) - \frac{1}{n}\sum_{m=1}^{n}\Phi(x_m)$$

$$= \Phi(x_i)\Phi(x_j)^T - (\frac{1}{n}\sum_{k=1}^{n}\Phi(x_k))\Phi(x_j)^T - \Phi(x_i)(\frac{1}{n}\sum_{m=1}^{n}\Phi(x_m)^T) + (\frac{1}{n}\sum_{k=1}^{n}\Phi(x_k))(\frac{1}{n}\sum_{m=1}^{n}\Phi(x_m)^T)$$

$$= K_{ij} - \frac{1}{n}K_{ij} - K_{ij}\frac{1}{n} + \frac{1}{n^2}K_{ij}$$

Thus, the data can be centered in the featured space by expressing the centered matrix in terms of the original kernel matrix K.

Finally, we solve the eigenvector problem

$$\hat{K}v = \lambda v$$

for eigenvalues λ and store the eigenvectors v in a matrix V. From [5], these eigenvectors are related to the eigenvectors of the covariance matrix (in the feature space) u by

$$u_i = \frac{1}{\sqrt{\lambda_i}}v_i$$

Then [5] gives that, for the used kernel function κ, the low-dimensional coordinates are given by

$$y_i = [\sum_{j=1}^{n} u_1^j \kappa(x_i, x_j), ..., \sum_{j=1}^{n} u_q^j \kappa(x_i, x_j)]$$

where u_1^j is the jth value in u_1.

1.2.1 Diffusion Maps

Diffusion mapping is another technique that handles data on a nonlinear manifold and uses full matrices. However, unlike kernel PCA, diffusion maps form a graph from the original data points and considers a random walk on the newly formed graph. A diffusion distance is then defined based on the transition probabilities, and by Proposition 1 of [6], the mapped low-dimensional data retains these diffusion distances (as good as possible) [1].

A notable benefit of this method is that it is, "based on integrating over all paths through the graph and is therefore impacted less by inaccurate or false connections made in the graph" [5].

To perform this algorithm, we start by calculating the weights of the edges in the graph of the data and storing them in an $n \times n$ matrix W. The weights are calculated using the Gaussian kernel function. Thus, each entry w_{ij} in W is given by

$$w_{ij} = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$$

To do this, the parameter σ must be chosen. As noted in 1, if σ, and therefore $2\sigma^2$, is small compared to $\|x_i - x_j\|^2$, this means that the entries in W will be approaching zero, while comparatively large values of σ mean the entries will approach one; a value that produces weights somewhere in between is generally desired, and some possible ways to accomplish this are presented in [8].

Next, a row-stochastic matrix P is formed, meaning the entries, denoted p_{ij}, of each row sum

[1] To see that the diffusion distance between data points is equal to the Euclidean distance between mapped points in the new space, [7] defines the diffusion distance as $\Delta_t(x_i, x_j)^2 = \sum_k |p_{ik}^t - p_{kj}^t|^2$ and then shows that $\Delta_t(x_i, x_j)^2 = \sum_k |p_{ik}^t - p_{kj}^t|^2 = \sum_z |p^t(x_i, z) - p^t(z, x_j)|^2 = \|\gamma_i - \gamma_j\|^2$ where γ_i is the column vector of transition probabilities at t time steps from x_i to x_n. In [7], Δ_t is denoted as D_t, but the change was made here to avoid mixing notation

to one, and then each row is divided by the respective row sum so that

$$p_{ij} = \frac{w_{ij}}{\sum_k w_{ik}}$$

Note that P^t gives the transition probabilities from one data point to another after t time steps.

If we form a diagonal matrix D, where the entries along the diagonal are the row sums of W, or $D_{ii} = \sum_j w_{ij}$, then P can be written as

$$P = D^{-1}W$$

Ultimately, we would like to once again solve an eigenvector problem. The following from [9] shows how to do this. Since eigenvectors corresponding to distinct eigenvalues are orthogonal for real, symmetric matrices, we symmetrize P by

$$\hat{P} = D^{\frac{1}{2}} P D^{-\frac{1}{2}}$$

Notice that since $P = D^{-1}W$, the above line is equal to $D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$. Additionally, for a particular entry of $\hat{P}$, we know that

$$\hat{p_{ij}} = \frac{w_{ij}}{d_{ii}}$$

and because each nonzero element (the diagonal entries) in D^{-1} is $\frac{1}{d_{ii}}$ (similarly, the diagonal entries in $D^{\frac{1}{2}}$ are given by $\sqrt{d_{ii}}$), we get that

$$\hat{p_{ij}} = p_{ij}\sqrt{\frac{d_{ii}}{d_{jj}}} = \frac{w_{ij}}{d_{ii}}\sqrt{\frac{d_{ii}}{d_{jj}}} = \frac{w_{ij}}{\sqrt{d_{ii}d_{jj}}}$$

Now, we solve for the eigenvectors u in

$$\hat{P}^{(t)}u = \lambda u$$

Then (as in [9] and Appendix 2 of [8]), compute $V = D^{-\frac{1}{2}}U$. The final low-dimensional coordinates are then computed by $Y = [\lambda_2 v_2, ..., \lambda_{q+1} v_{q+1}]$ because the first eigenvalue is equal to one and corresponds to an eigenvector with all entries the same (meaning this eigenvalue is trivial) and is subsequently ignored [5]. In practice, before removing the first column, the matrix Y is sometimes divided by the entry $y_{1,1}$ (which merely rescales the new coordinates) since the first column will be constant and corresponds to the eigenvalue $\lambda_1 = 1$ [5] [2].The first column is subsequently ignored and the next q (rescaled) eigenvectors are kept and give the new representation [5].

1.3 Nonlinear Sparse Matrix Methods

The methods of kernel PCA and diffusion mapping both dealt with full matrices to achieve a lower-dimesional representation of data on a nonlinear manifold. The next two methods, local linear embedding and Laplacian eigenmaps, solve eigenvector problems involving sparse matrices.

1.3.1 Local Linear Embedding

As before, suppose X is a data matrix of size $n \times p$ where n is the number of data points and p is the number of variables measured. Local linear embedding (LLE) maps the data points to a lower dimensional space by rewriting each data point as a linear combination of different, nearby data points in the original space, preserving the coefficients of each linear combination (called reconstruction weights), and then reforming the data points from linear combinations of the lower-dimensional points and reconstruction weights.

The key assumption behind this method is that each original data point and its nearest neighbors exist on a locally linear section of the true underlying manifold (which is itself assumed to be smooth) [10]. The algorithm from [10] is outlined below.

This method begins by selecting the nearest neighbors of each data point, which can be done in

[2]This can be done in more than one way; [8] cites the Matlab function eig() to acquire the eigenvectors while the dimension reduction toolbox created by Laurens van der Maaten (link given in the next chapter) uses the function svd().

two ways. The first is to consider a ball of radius ε centered at the given data point x_i and writing that point as a linear combination of all other points within that ε-ball. The second way allows the user to set an integer value for k and then writing x_i as a linear combination of the k nearest data points.

Suppose the latter approach mentioned above is taken. Once the k nearest neighbors have been identified, the cost function

$$\Phi_1(w) = \sum_i \|x_i - \sum_j^k w_{ij} x_j\|^2$$

gives the sum of the square differences between the data points and the respective reconstructed representations. Subject to two constraints, minimizing this first cost function allows the reconstruction weights w_{ij} to be calculated. The first constraint is that each x_i is reconstructed from only the k-nearest neighbors and every other point is ignored, i.e. for points that are not a nearest neighbor, $w_{ij} = 0$. The other constraint is that $\sum_j w_{ij} = 1$ (the rows of W sum to one) [3]. Consequently, the reconstruction weights are retained through a linear map to a lower dimensional space [4]. Since the w_{ij} (with the nearest neighbors) reconstruct x_i in the high dimensional space and are retained in the low dimensional space, the low dimensional representation of an arbitrary data point, y_i, will also be reconstructed using these weights and the k-nearest neighbors of y_i.

To actually calculate the w_{ij}, it is shown in [10] (Appendix A) that for a data point x_i, weights w_{ij} that sum to one[5], neighboring points α_j, and local covariance matrix defined as

$$C_{jl} = (x_i - \alpha_j) \cdot (x_i - \alpha_l)$$

the reconstruction error for x_i can be expressed as

$$E = \sum_i |x_i - \sum_j^k w_{ij} \alpha_j|^2 = |\sum_j^k w_{ij}(x_i - \alpha_j)|^2$$

[3] The matrix W satisfies the conditions to be a stochastic matrix and can be viewed as such, though [10] does not discuss W in such a context

[4] The weights are invariant to rotation, rescaling (by the first constraint), and translation (by the second constraint) [10].

[5] [10] notes that a Lagrange multiplier is used to enforce the condition that $\sum_j w_{ij} = 1$.

$$= \sum_{jl} w_j w_l C_{jl}$$

where the fact that the weights sum to one is used to achieve the second equality. The weights are then given in terms of inverse local covariance matrices

$$w_{ij} = \frac{\sum_l C_{jl}^{-1}}{\sum_{mr} C_{mr^{-1}}}$$

However, it is pointed out in [10] that in practice, the system of linear equations given by $\sum_j C_{jl} w_l = 1$ can be solved to minimize this error, and rescaling the weights so that the rows of W sum to one. In practice, [10] does this by dividing each row by the row sum.

To construct the y_i, the following (second) cost function is minimized:

$$\Phi_2(y) = \sum_i \|y_i - \sum_j^k w_{ij} y_j\|^2$$

The first cost function, Φ_1, is minimized by finding weights w_{ij} for each nearest neighbor so that the sum of the reconstruction errors is as small as possible; this cost function penalizes “poor" representation of all x_i by the respective linear combinations. Given the weights, then, the second cost function is minimized by finding the k-nearest neighbors of y_i in low dimensional space. This cost function is high if, given the weights w_{ij}, the neighbors of y_i construct a poor representation of y_i.

However, instead of optimizing, an eigenvector problem may be solved, subject to a couple of constraints. For the jth column of Y, the matrix containing the new coordinates, $\|y^{(j)}\|^2 = 1, \forall j$, which eliminates the (trivial) solution with eigenvalue zero and eigenvector whose entries are all the same [5]. Another way of viewing this, given in [10], is

$$\frac{1}{n} \sum_i y_i y_i^T = I$$

The second constraint imposed is

$$\sum_i y_i = \vec{0}$$

whose purpose is to, "remove the degree of freedom of translating the coordinates of y_i by a constant displacement" [10] [6].

Now, from [10] (Appendix B), consider the matrix M whose entries are given by

$$m_{ij} = \mathbf{1}_{ij} - w_{ij} - w_{ji} + \sum_k w_{ki} w_{kj}$$

where $\mathbf{1}$ is equal to one if $i = j$ and zero if $i \neq j$. Then we have that

$$\Phi_2(Y) = \sum_{ij} m_{ij}(y_i \cdot y_j)$$

According to [10], the eigenvectors of M give the optimal embedding. To get the desired eigenvectors, [10] highlights the fact that M can be written as

$$M = (I - W)^T (I - W)$$

and from [5],

$$\Phi_2(Y) = (Y - WY)^2 = (Y - WY)(Y - WY)$$

$$= (I - W)Y \cdot (I - W)Y = Y^T (I - W)^T (I - W)Y$$

Then the lower-dimensional coordinates for the y_i come from the following eigenvectors v corresponding the the smallest eigenvalues (where Φ_2 is minimized by this solution [5]):

$$Mv = \lambda v$$

Here, the bottom eigenvector corresponds to the eigenvalue of zero and is ignored; the next q

[6] In [10], the cost function Φ_2 is invariant to rotation and rescaling which prevents a loss of generality from imposing this condition on the covariance

bottom eigenvectors v of M are kept.

1.3.2 Laplacian Eigenmaps

Laplacian Eigenmaps is the last technique explored and is similar to LLE. A graph of the original data points is formed, the nearest neighbors of each data point x_i are identified, the edges of the neighbors are weighted, and an eigenvector problem is solved using matrices based on the weight matrix W. The following summarizes the algorithm in [11].

As in LLE, the first step is to construct a graph of the data points and find the nearest neighbors of each data point. This may be done in the same two ways as before: setting a value for a parameter ε and letting the neighbors of x_i be all other data points within a distance of ε or less from x_i, or setting an integer value for k and choosing the k closest points to x_i. Again, suppose the latter method is chosen.

Next, the weight matrix W must be constructed. Once again, if x_i and x_j are not neighbors, then $w_{ij} = 0$. If x_i and x_j are connected, then w_{ij} may be computed in one of two ways. The first is to use the Gaussian kernel (sometimes referred to as the heat kernel). The second way is to set $w_{ij} = 1$ if two data points are neighbors.

Once W is formed (which is symmetric), the diagonal matrix D is formed by taking the column (or row) sums of W and making these values the diagonal entries of D, so that $d_{ii} = \sum_j w_{ij} = \sum_i w_{ij}$. The graph Laplacian 12 (in matrix form) is then calculated by $L = D - W$.

To keep the local structure as good as possible in the new space [5], the following cost function is minimized:

$$\Psi(Y) = \sum_{ij} (y_i - y_j)^2 w_{ij}$$

The cost should be high if x_i and x_j are close but y_i and y_j end up far away from each other. Note that a sense of order is present in the cost function Ψ, as the difference between a point x_i and its first neighbor receives more weight than the difference between it and its second neighbor, etc. [5].

Fortunately, this minimization problem can be viewed as an eigenvector problem [5] because

by expanding Ψ and distributing the w_{ij} term, one sees that

$$\Psi(Y) = \sum_{ij}(y_i - y_j)^2 w_{ij}$$

$$= \sum_{ij}(y_i^2 + y_j^2 - 2y_i y_j^T) w_{ij}$$

$$= \sum_{ij} y_i^2 w_{ij} + y_j^2 w_{ij} - 2y_i y_j^T w_{ij}$$

and since $d_{ii} = \sum_j w_{ij} = \sum_i w_{ij}$,

$$= \sum_i y_i^2 d_{ii} + \sum_j y_j^2 d_{jj} - \sum_{ij} 2y_i y_j^T w_{ij}$$

$$= 2Y^T DY - 2Y^T WY$$

$$= 2Y^T LY$$

Thus, $\Psi(Y) \propto Y^T LY$ under the constraint that $Y^T DY = I$ where I is the $n \times n$ identity. The eigenvector problem to solve, then, is

$$Lv = \lambda Dv$$

for the eigenvectors v. The smallest eigenvalues and corresponding eigenvectors yield the minimizing vectors v_i. However, $\lambda = 0$ is an eigenvalue and corresponds to the trivial eigenvector of all ones, so the next q smallest eigenvectors are used to form the lower-dimensional representation Y, which contains the new coordinates [5]. For more details on this, see [11].

Chapter 2

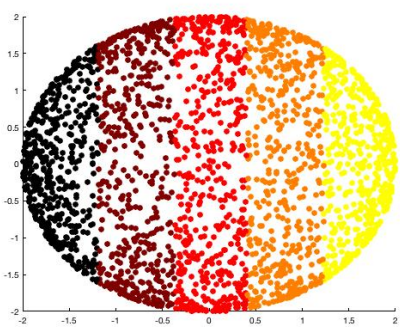
Results and Comparison

In this section, the methods previously discussed are demonstrated by applying them to various data sets designed to illuminate the strength(s) of each method. Additionally, some benefits and drawbacks of the methods are discussed. The methods were implemented in Matlab and are based on the drtoolbox created by Laurens van der Maaten which can be downloaded at `https://lvdmaaten.github.io/drtoolbox/`. The performance metrics of trustworthiness and continuity are calculated [5], as well as the average of these two metrics. To the author's knowledge, this average has not been used before. The LLE code is based on that of [10] and [5]. The codes used to generate the images were not optimized for any particular purpose (run time, memory use, etc.). Some notes about the scripts used are also mentioned for each method, and some applications each method has been applied to are mentioned. Finally, results from some of the techniques performing on Persyst EEG data are presented.

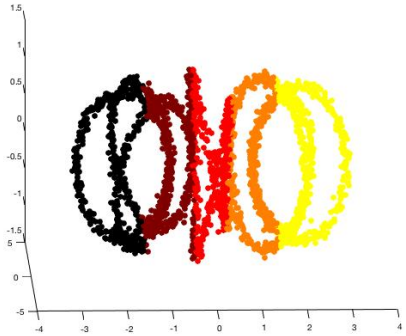
2.0.1 The Artificial Data Sets

Algorithms for the methods presented were each performed once on data with three different structures: a sphere, helix, and Swiss roll. These manifolds are shown below and were each constructed from 2500 data points.

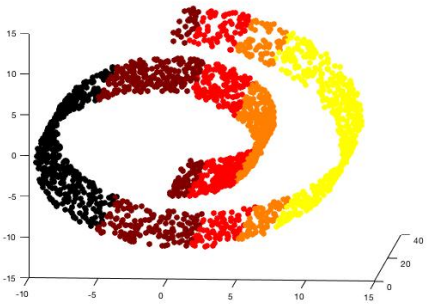
The Swiss roll data set is a classic example on which dimensionality reduction techniques are (and have been) performed on. Testing them on this data set shows how well techniques handle data lying on a low-dimensional manifold isometric to Euclidean space, while the helix data set tests for performance on data lying on a low-dimensional manifold not isometric to Euclidean



(a) Sphere



(b) Helix



(c) Swiss roll

Figure 2.1: Manifolds of data sets

space [5]. The following analysis, to author's knowledge, has not been performed on a sphere at the time of writing. Each data set has three variables and each method aims to reduce to two dimensions.

2.0.2 Implementation of Techniques

In the following sections, some notes about the implementation of the methods are given, as well as some known types of applications for each method. For each method, the data sets consist of points in $\mathbb{R}^3$ and are reduced to two dimensions. The choice of retaining two dimensions comes from the goal of actually producing a representation with fewer dimensions.

The metrics of trustworthiness, continuity, and average of these values are also calculated for each method. The trustworthiness is defined as

$$T(K) = 1 - \frac{2}{nK(2n-3K-1)} \sum_{i=1}^{n} \sum_{j \in U_i^K} (r(i,j) - K)$$

which describes, "the extent to which a neighborhood of points in the lower-dimensional space correctly corresponds to a neighborhood around the same point in the original space" [12]. Alternatively, [5] describes this as, "a measure of the proportion of points too close together in the low-dimensional space." Here, n is the number of data points, K is the size of the neighborhood, or number of points that constitute a neighborhood (and not necessarily equal to k in LLE and Laplacian eigenmaps), $r(i,j)$ is called the "rank of the low-dimensional data point j according to the pairwise distances between the low-dimensional data points" [5]. To clarify, this "rank" may be viewed as that of a contest of which points are nearest to a data point. For example, a data point with rank 8 is the eighth closest point to the initial point (including the initial point, which of course has distance zero to itself). Finally, U_i^K is the set of points that are nearest neighbors of a particular ith data point in the low-dimensional space but *not* neighbors of the same ith data point in the high-dimensional space.

The continuity metric can be thought of as a sort of converse to trustworthiness; it describes,

Table 2.1: Trustworthiness

	PCA	Poly.	Gauss	DM	LLE	LEM
Helix	.9427	.6320	.9983	.9986	.8564	1.00
Swiss	.8869	.8873	.6272	.9988	.7226	.8833
Sphere	.8523	.5256	.8517	.9998	.6920	.8682

Table 2.2: Continuity

	PCA	Poly.	Gauss	DM	LLE	LEM
Helix	.9992	.9975	.9988	.9998	.9971	1.00
Swiss	.9977	.9921	.8028	.9984	.9665	.9877
Sphere	.9979	.9857	.9982	.9999	.9546	.9973

"the extent to which the neighborhood of every point in the high-dimension space is kept the same after mapping to the lower-dimensional space" [12]. Similar to trustworthiness, $s(i, j)$ is the rank of a high-dimensional data point j according to high-dimensional pairwise distances and V_i^K is the set of points that are nearest neighbors in the original space but not in the low-dimensional space. The formula is given by

$$C(K) = 1 - \frac{2}{nK(2n - 3K - 1)} \sum_{i=1}^{n} \sum_{j \in V_i^K} (s(i, j) - K)$$

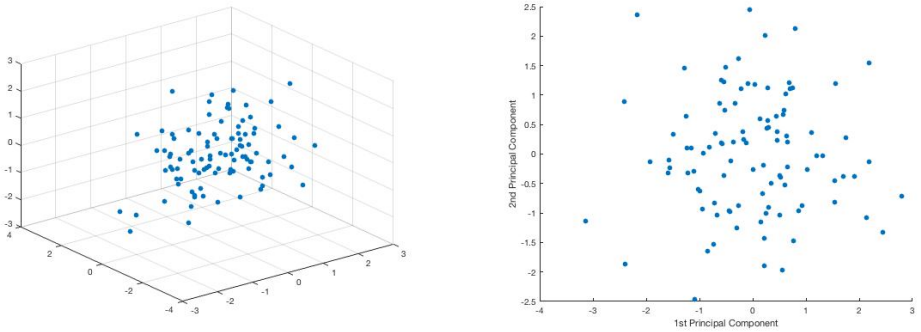
Note that for both of these metrics, the double sum is always positive since each term is positive (the rank must always be larger than K). Note also that these metrics produce values between 0 and 1 and that the higher the result, the better the performance. Because both metrics are interpreted the same way, their average, denoted by τ, is also calculated, providing an "overall performance" metric with regard to the nearest neighbors of each point being the same in the starting and final dimensions. Here, the performance of each method was calculated once using $K = 20$. The results of each method on these data sets are presented in the tables below. For techniques with parameters to set, the default values (mentioned in their respective sections) were used.

Table 2.3: Average, τ

	PCA	Poly.	Gauss	DM	LLE	LEM
Helix	.9710	.8148	.9986	.9992	.9268	1.00
Swiss	.9423	.9397	.7150	.9986	.8446	.9355
Sphere	.9251	.7557	.9250	.9999	.8233	.9328

2.0.2.1 PCA

For the purpose of a visual demonstration of dimension reduction, Figure 2 shows what dimension reduction on data from the standard normal distribution looks like using PCA.



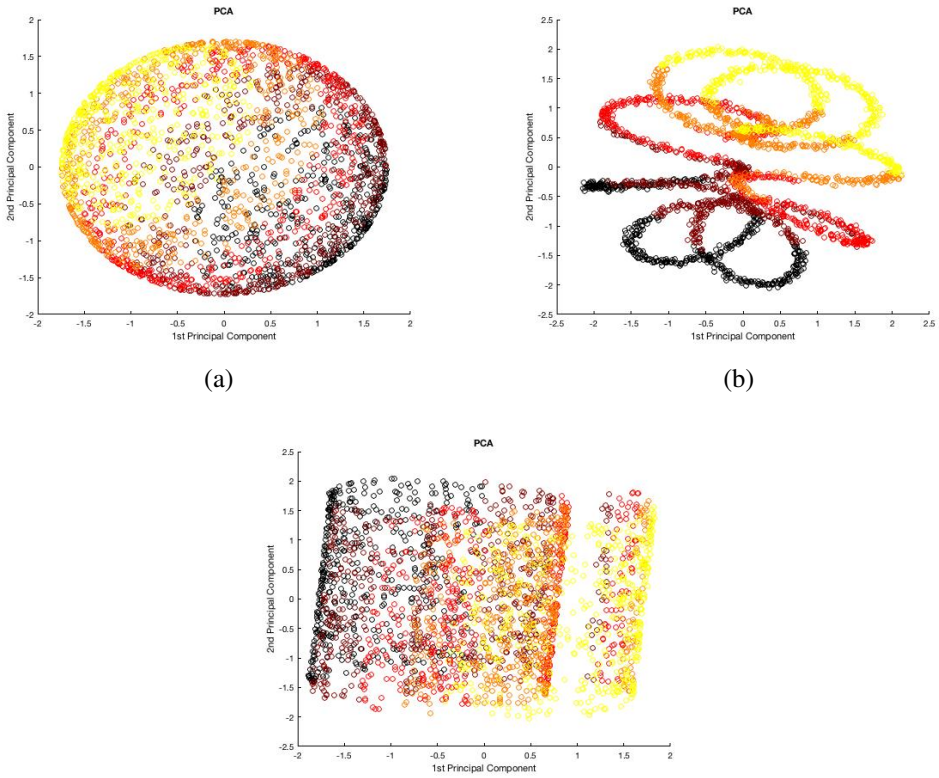
(a) 100 data points generated from the the standard normal distribution in three dimensions

(b) PCA reducing to two dimensions

Figure 2.2: PCA performed on data originally in $\mathbb{R}^3$

Figure 2.3 in the next section shows what PCA looks like reducing data from three dimensions to two.

It was shown earlier that PCA with column centered data is equivalent to SVD. This equivalency is worth noting because Matlab contains a built in function, pca(), that performs PCA. However, the mechanics of this function consist of centering the columns and then doing a singular value decomposition. This function was used to run the PCA algorithm.



(a) (b)

(c)

Figure 2.3: PCA with two components

Figure 2.3 shows how PCA performs on the three data sets. From this, one sees that PCA essentially flattens the manifolds. Moreover, it does not appear to identify groupings or clusters of points in any way more meaningful than the original representation of the data in Fig. 1. While unsurprising, this may further motivate the utility of other methods.

However, referring to the performance tables, PCA may seem to do well (perhaps justifying its widespread use in practice) but does not perform best for any data set; although, its trustworthiness is comparatively good.

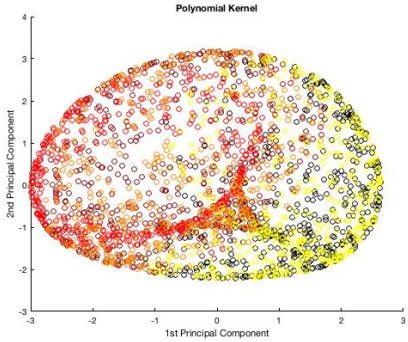
PCA can be, and is, used for many applications, including but not limited to: seismic series analysis [13], facial recognition [14], and analyzing the size of fossil teeth [1].

2.0.2.2 Kernel PCA

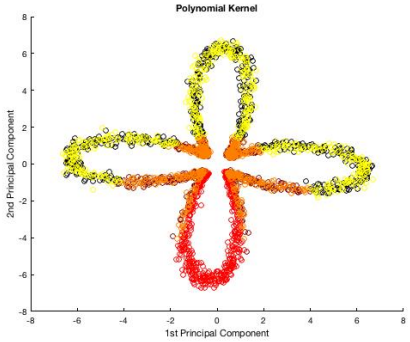
Kernel PCA has the obvious benefit of being designed to handle nonlinear data while maintaining an algorithm that closely resembles traditional PCA. However, there are two primary drawbacks to kPCA: the kernel matrix has size $n \times n$, so a large number of data points can mean the kernel matrix is large, and kPCA retains large pairwise distances between data points (as opposed to LLE and Laplacian Eigenmaps, which retain small pairwise distances) [5].

The first step in kPCA is choosing a kernel function, which may be a benefit and hindrance. There is nothing that guarantees all kernel functions will perform equally well, even with well-chosen values for parameters of the given kernel function. Subsequently, one may need to set values for the chosen kernel function, which may not be easy to do. In the case of the polynomial kernel, the parameter c was set to 1 by default, and the power p was set to 2, yielding a quadratic polynomial. The Gaussian kernel parameter σ can be challenging to estimate and is talked about in more detail in the next section. By default, $\sigma = 1$.

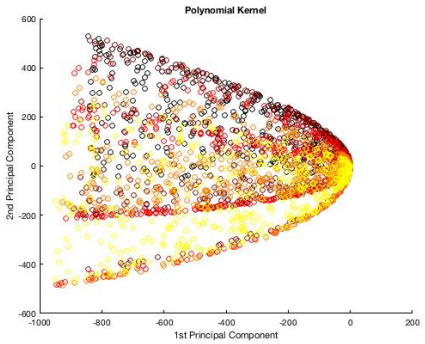
Below are the outputs for the polynomial and Gaussian kernels on the sphere, helix, and Swiss roll data sets, respectively.



(a) Sphere

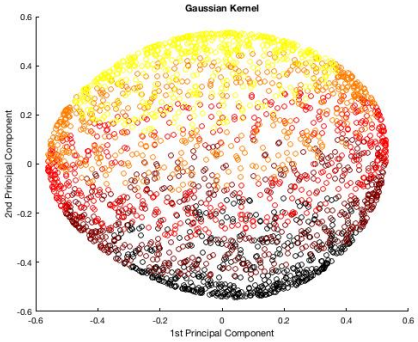


(b) Helix

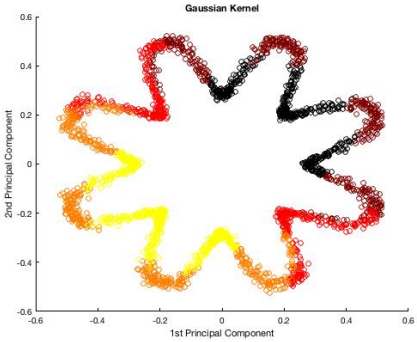


(c) Swiss roll

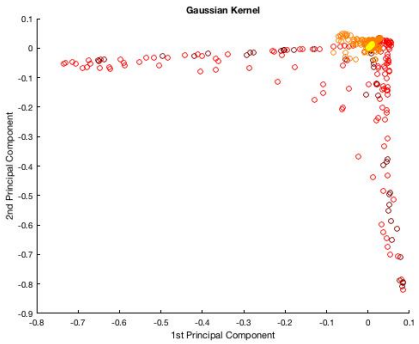
Figure 2.4: kPCA using a polynomial kernel



(a) Sphere



(b) Helix



(c) Swiss roll

Figure 2.5: kPCA using a Gaussian kernel

Here, some clustering can be seen on the Swiss roll data set with both kernels, but many points are still ungrouped. Indeed, [5] reports that “techniques that do not employ neighborhood graphs perform poorly on data sets like the Swiss roll."

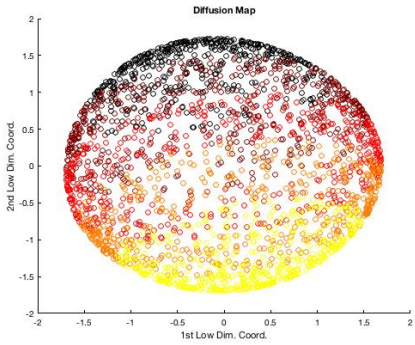
The polynomial kernel outperformed the Gaussian kernel on the Swiss roll, though overall the polynomial kernel performed poorly compared to the other methods. In particular, this choice of kernel function was the least trustworthy for both the helix and sphere.

Barring the choice of a linear kernel, kPCA has been applied to face recognition [15].

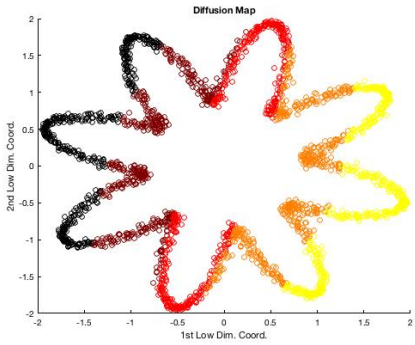
2.0.2.3 Diffusion Mapping

Diffusion maps begin by forming a Markov chain, so one parameter that can be set is the number of time steps t. By default, this algorithm used $t = 1$, although as an intermediate step, one could run the Markov matrix forward any number of time steps algorithm.

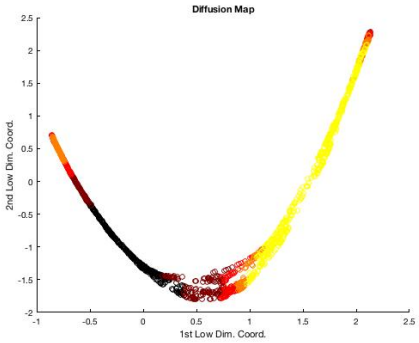
It is also worth noting that, in the drtoolbox, some pre-processing of the data is done with this method. In particular, the data is manipulated so that every value is in $[0,1]$. This is done by subtracting the minimum value from each entry and then dividing each entry by the maximum value. While the shifting of data into $[0,1]$ is not prohibited, and indeed, some form of pre-processing may be worthwhile or necessary, the algorithm does not explicitly list it as a step. Thus, the images in this section did not use any pre-processing.



(a) Sphere



(b) Helix



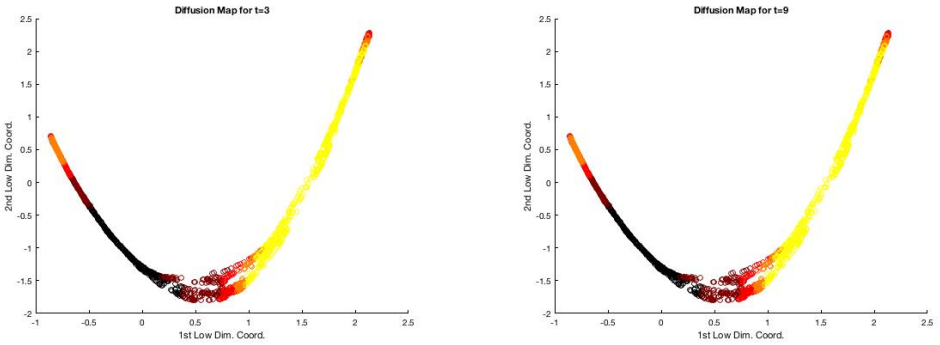
(c) Swiss roll

Figure 2.6: Diffusion Mapping

Figure 2.6 shows the diffusion mapping algorithm performed using $t = 1$ and $\sigma = 1$. Again, the sphere and helix are more or less flattened. However, even though this method has been found to not perform as well as LLE and Laplacian eigenmaps [5], some clustering can be seen on the Swiss roll. On the whole, this technique performed consistently well on all three data sets.

Now, varying the parameter t is examined for the case of the Swiss roll. Moving the stochastic matirx forward three and nine time steps produces the images in Figure 2.7. It does not appear

that evolving the stochastic matrix forward had any significant effect on this particular data set. However, [6] points out that as t increases, the number of significant eigenvalues decreases. Also in [6], the authors discuss the applicability to image processing.



(a) Three time steps (b) nine time steps

Figure 2.7: Diffusion mapping with different time steps

A more difficult task than choosing the parameter t, perhaps, is choosing σ. There are several options listed in [9], one of which is using the median of the Euclidean distances between all data points,

$$med\{\|x_i - x_j\|^2\}$$

and setting this equal to $2\sigma^2$. Figure 2.8 shows what happens when the median is used.

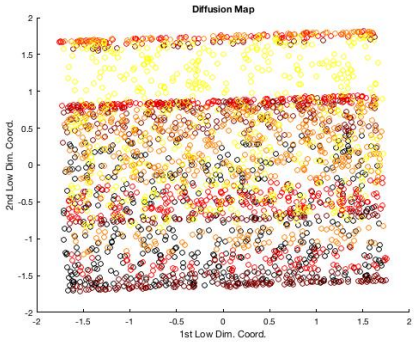


Figure 2.8: Diffusion Map with median for Gaussian kernel parameter

Clearly, the choice of σ has an effect on the output. Visually, one sees that this choice of σ causes the diffusion map algorithm to produce an image that resembles PCA. From this alone,

using the median seems to not be a good choice. However, this resulted in a value of 1 for both trustworthiness and continuity, which seems to highlight the fact that these metrics alone may not necessarily indicate how well a method performs. These metrics only show what proportion of points close together in the original and lower-dimensional spaces were also close together in the other space [1].

Additionally, [9] describes a large σ as causing a larger diffusion distance and a small σ indicates a scarcely connected graph of the data points (if one imagines representing the Markov chain in graph form). Furthermore, [7] says that for manifolds that do not have very high dimension to begin with and data that seem close together, a smaller value for σ is more appropriate. The Swiss roll data set generated appears to fit this description, and indeed, the median value was approximately 292, implying that sigma was approximately 12.1 (using the equation median = $2\sigma^2$). So while the difference in values for σ may not seem like much, the value in the denominator increased from 2 (using $\sigma = 1$) to just above 292. Thus, the median value was likely too high. In general, σ should be carefully chosen [6].

Diffusion maps have been applied to shape matching [17] and, recently, analyzing EEG data for preseizure conditions [18].

2.0.2.4 LLE

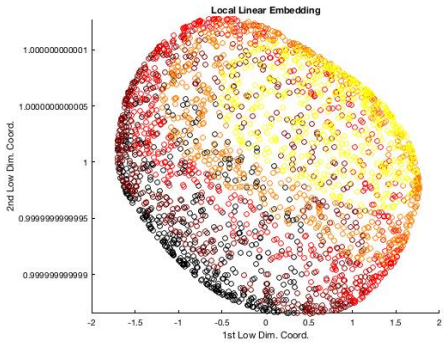
LLE works on the idea of being able to express each data point as a linear combination of its nearest neighbors, all of which are assumed to lie on a locally linear hyperplane of the original manifold. Ultimately, LLE aims to preserve only local properties of the given data points. Consequently, this method is also robust to short-circuiting [5]. The images here were generated from code based on that of [10], whose code can be found at `https://cs.nyu.edu/~roweis/lle/code.html` [2].

Perhaps the most obvious drawback is that the assumption of local linearity may not be valid. This can be easy to verify when the true underlying structure is known, but in practice such infor-

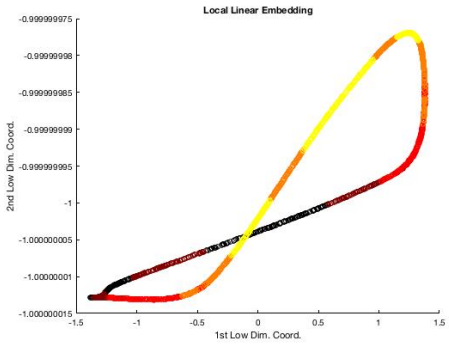
[1] In [5], another performance metric, the generalization error of 1-nearest neighbor classifiers [16], was used, and [5] reports that diffusion mapping had one of the worst performances on the Swiss roll with respect to this metric.

[2] Note that the founders of this technique consider X to be a $p \times n$ matrix. Thus, if one wanted to stay consistent with the notation of this paper and use code from the above link, the data matrix will need to be transposed first.

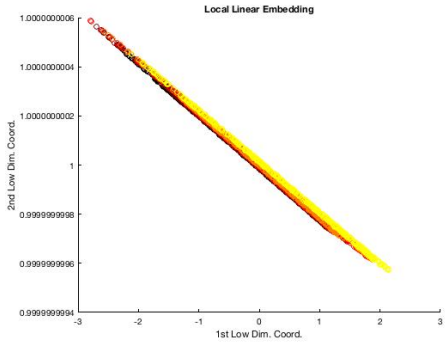
mation may be unattainable. Additionally, the choice of how to find neighbors for each point may impact the performance; one either must choose ε or k so that a reasonable number of neighbors are found for each point. It is easy to control this using k, but if using an ε-ball, one should check to see how many neighbors each point acquires in the algorithm-it may not be the same number for each data point. Lastly, using the e-ball method is considered more geometrically intuitive, but k-nearest neighbors is easier to implement [10]. LLE also seems to have difficulty performing on manifolds containing holes, and, because of the (local) covariance restriction, tends to collapse data close together in the low dimensional space [5].



(a) Sphere



(b) Helix



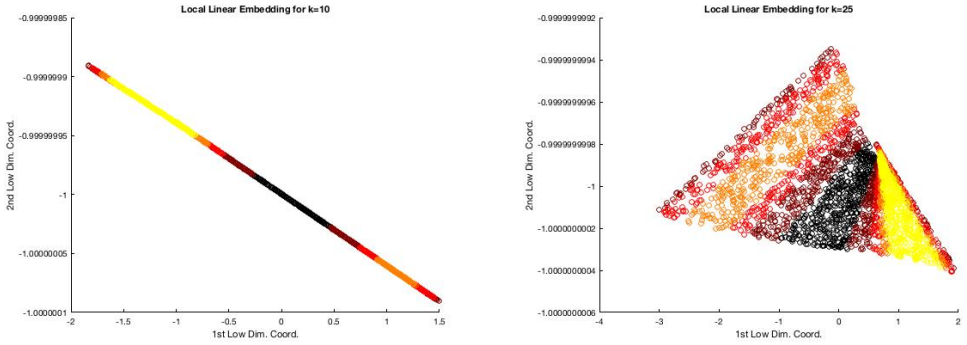
(c) Swiss roll

Figure 2.9: Local Linear Embedding with $k = 50$

In terms of the performance metrics, LLE had a tendency to be one of the worst with respect to trustworthiness. Comparing LLE to Laplacian eigenmaps, the other method based on nearest neighbors and using a sparse matrix, the latter performs markedly better.

The only parameter to vary in the specific implementation of this method is k. LLE was per-

formed on the Swiss roll data set using 10 and 25 nearest neighbors as well. The results are displayed in Figure 2.10.



(a) k = 10 (b) k = 25

Figure 2.10: Local Linear Embedding with varied k

The image in Fig. 9 for the Swiss roll used 50 neighbors (default). However, even with $k = 10$, one can see that the Swiss roll is 'unrolled' and the perspective is from a cross-section of that plane. For $k = 10$, LLE performed with the following results: $T(K) = .8017$, $C(K) = .9850$, and $\tau = .8933$, and for $k = 25$, the results were $T(K) = .7962$, $C(K) = .9849$, and $\tau = .8905$. Thus, LLE appears to have a noticeable drop in performance when going from 25 to 50 neighbors.

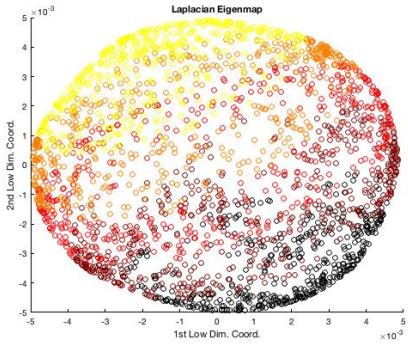
LLE has been found to work well in applications like superresolution [19] and sound source localization [20].

2.0.2.5 Laplacian Eigenmaps

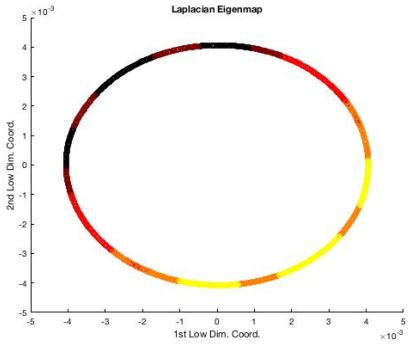
Laplacian eigenmaps is similar to LLE in that neighborhoods of every data point are formed, but makes use of the Gaussian kernel like a couple of other methods presented here. Laplacian eigenmaps code different from that which was used for the images here can be downloaded at `http://web.cse.ohio-state.edu/~belkin.8/algorithms/algorithms.html`.

Due to the resemblance of LLE, Laplacian eigenmaps have similar advantages and disadvantages. Note that both sparse methods produce a trivial solution that must be excluded [5]. The number of neighbors was varied in a manner identical to LLE for the Swiss roll. Also, since the

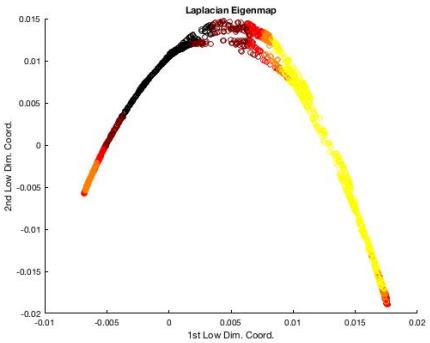
Gaussian kernel is present, the algorithm was also run using the median of all nonzero values in the sparse matrix W for σ and default value of $k = 50$.



(a) Sphere



(b) Helix

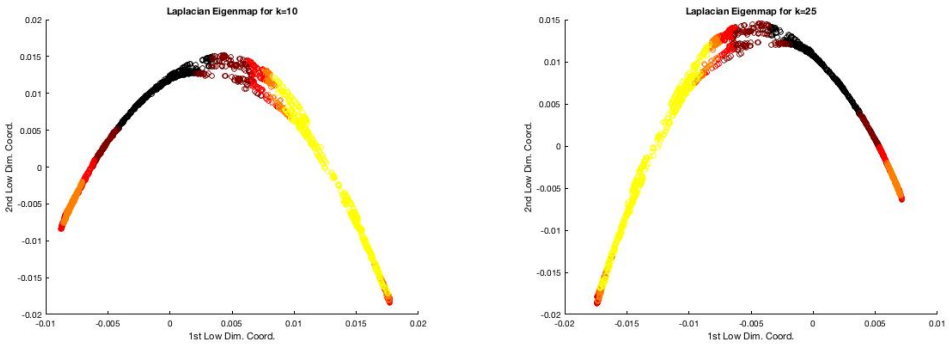


(c) Swiss roll

Figure 2.11: Laplacian eigenmaps with $k = 50$

This method performed better than any other on the helix data set, suffered (relatively) on the Swiss roll, and had the second highest trustworthiness for the sphere, but lacked in continuity.

Again, varying the value of k produces



(a) k = 10 (b) k = 25

Figure 2.12: Laplacian eigenmap with varied k

For $k = 10$, the performance results were $T(K) = .9183$, $C(K) = .9881$, and $\tau = .9532$, and for $k = 25$, the results were $C(K) = .9039$, $T(K) = .9873$, and $\tau = .9456$, indicating that a larger neighborhood size may not necessarily be better.

If σ is varied, however, the result is different.

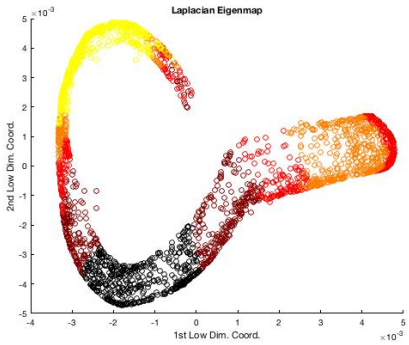


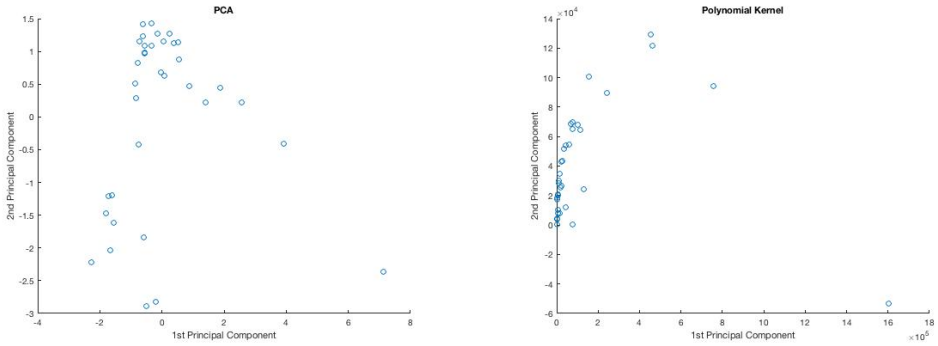
Figure 2.13: Laplacian eigenmap with median for Gaussian kernel parameter

The actual value for σ was approximately 12.1 (the value for the denominator was, again, just over 292) compared to the default value of $\sigma = 1$. This time, values of $T(K) = .9522$, $C(K) = .9940$, and $\tau = .9731$ are reported. Thus, Laplacian eigenmaps seems to perform slightly better overall with the median when using $k = 50$ neighbors.

Laplacian eigenmaps have been successfully applied in areas such as facial recognition [21] and fMRI data analysis [22].

2.0.3 Persyst EEG Data Set

This data set consists of data collected during a study of 46 participants, though some participants did not have this particular data available and were therefore removed from the sample, leaving a sample size of $n = 35$. The data set originally measured six variables and for each method, the set was reduced to two dimensions. For methods that require parameters to be chosen, the default values were used. The performance metrics were calculated using $K = 12$. The two-dimensional representations and performance results are given below. For Laplacian eigenmaps, the appropriate function from the drtoolbox mentioned at the beginning of chapter three was used. Results for the Gaussian kernel were not obtained for this data.



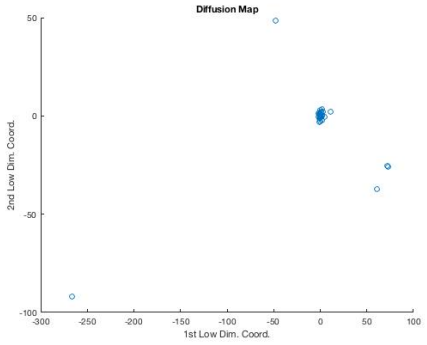
(a) PCA (b) Polynomial kernel PCA

Figure 2.14: PCA and Polynomial kPCA on Persyst EEG data

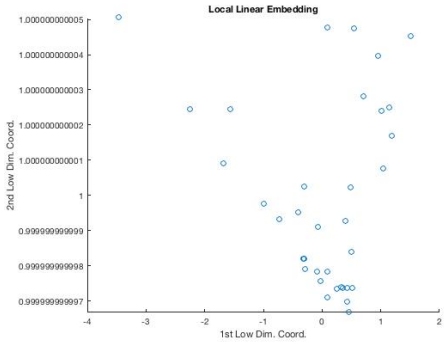
Just from examining the graphical results, PCA appears to have at least a semblance of clustering. Of the six variables measured, the first principal component primarily represents the first three, which are average number of (EEG) spikes per minute, average number of spikes per hour, and maximum number of spikes per hour. The second principal component primarily represents the total hours analyzed and maximum PIH end. For two principal components, the proportion of variance retained was 82.48%.

Table 2.4: Persyst EEG Performance

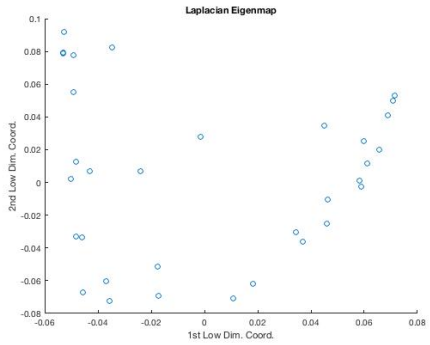
	PCA	Poly.	DM	LLE	LEM
T(K)	.9698	1.00	.8160	.9537	.9391
C(K)	.9593	.9993	.8139	1.00	1.00
τ	.9646	.9987	.8149	.9779	.9722



(a) Diffusion Mapping



(b) LLE



(c) Laplacian eigenmap

Figure 2.15: Diffusion maps, LLE, and Laplacian eigenmaps on Persyst EEG data

Some clustering can be seen in some of these other methods too, namely diffusion mapping and some in LLE. Each method was analyzed with respect to the same performance metrics as before. The results are in the table below.

From these results, one sees that Laplacian eigenmaps seems to have performed well, similar to the artificial data sets. However, the polynomial kernel and LLE also performed well, and diffusion maps, in contrast to the artificial data sets, performed the worst here. It is worth noting that, due to its widespread use, PCA seems to ahve performed considerably well here. However, these results

may change if the parameters for the methods are varied in ways mentioned previously. The value for K, if changed, could also produce different results. Indeed, these are only preliminary results; more investigation and analysis should be conducted for a stronger conclusion about the performance of these methods on this particular data set.

Chapter 3

Conclusion

This paper presented several methods of dimensionality reduction and enumerated the steps of each algorithm so that they may be more easily implemented in a programming language. Distinctions between the techniques were made by categorizing them into full and sparse techniques with key features of each method highlighted. In addition to explaining the methods, each technique was performed on three data sets and the 2D results were provided and discussed. Where appropriate, methods were run repeatedly with different values for parameters of the respective method. Notably, the number of time steps in diffusion mapping, the number of neighbors in LLE and Laplacian eigenmaps, and σ in methods involving the Gaussian kernel function. Given the observed differences in outputs for these methods, the relationship between these parameters themselves (where applicable), as well as the final output, should be investigated further. In particular, which value(s) for σ yields the best output, the most consistent output, and whether or not this is dependent on the specific application.

For nearest neighbor methods, using an ε-ball instead of k-nearest neighbors would be interesting to explore further. To the author's knowledge, the case where the original data points might have different numbers of neighbors and the effect of this on a 2D representation (or more generally, lower dimensinal representation) has not been analyzed, namely whether or not this affects how well certain points are reconstructed in the new space. Extensions of LLE and Laplacian eigenmaps are presented in [23]. It would also be worthwhile investigating what the relationship is between the size of the neighborhood and the performance of these methods for a particular application/type of data.

The metrics of trustworthiness and continuity should also be tested more. The results here

indicate that, overall, diffusion mapping is the most reliable way of doing dimension reduction on these sets. This conclusion is despite what was mentioned earlier, namely that neighborhood-based techniques do better on data sets like the Swiss roll. However, Laplacian eigenmaps also performed well, and these performance values were not calculated multiple times as in 12. A different value of $K = 20$ was also used instead of $K = 12$. Furthermore, the average of trustworthiness and continuity, τ, was introduced. Some methods have τ values that are close, such as PCA, polynomial kPCA, and Laplacian eigenmaps on the Swiss roll, but whether or not these small differences are actually significant should be investigated. The value τ is susceptible to the shortcomings of any average value, but it may be a starting point from which a modification can be made so that the performance of a particular method on a given data set can be summarized in one number. It was also seen that trustworthiness and continuity may not be reliable indicators of performance if they are all that as considered, since diffusion mapping using the median had excellent performance based on these proportions, but the visual representation did not appear to provide any meaningful clustering of the data points.

Additionally, some methods were used on a real data set consisting of Persyst EEG data. The results presented here are preliminary. In particular, the performance of each method on the data set, while they may be indicative, should be explored further in ways previously mentioned.

Perhaps the most important future direction for this project, though, is to analyze a large, real data set. EEG data was graciously provided by the Laboratory of Neuro Imaging at the USC Stevens Institute of Neuroimaging and Informatics in the Keck School of Medicine of USC as part of the Epilepsy Bioinformatics Study for Antiepileptogenic Therapy. However, at the time of writing, the author did not have a computer available (or access to one) that could adequately handle the very large data set(s). From the data received, with access to capable equipment and with permission, the first step would be to perform dimension reduction on the electrical signals recorded from electrodes, where the electrodes would be taken to be the data points. This would establish some type of spatial relation with the electrical brain activity recorded. Finally, as data is currently collected in multiple ways, e.g. images, numerical measurements, etc., comparing the

performance of dimension reduction methods on MRI data to EEG data, for example, and seeing if similar descriptions of the data are formed by each method on each type of data set would be of particular interest.

References

[1] I. Jolliffe and J. Cadima, "Principal Component Analysis: A Review and Recent Developments," *Phil. Trans. R. Soc. A*, 2016.

[2] B. Schölkopf, A. Smola, and K.-R. Müller, "Kernel Principal Component Analysis." Springer, Berlin, Heidelberg, 1997, pp. 583–588.

[3] M. Welling, "Kernel Principal Component Analysis," Department of Computer Science, University of Toronto, Tech. Rep.

[4] [Online]. Available: http://research.cs.tamu.edu/prism/lectures/pr/pr_l28.pdf

[5] L. van der Maaten, E. Postma, and J. van den Herik, "Dimensionality Reduction: A Comparative Review," TiCC, Tilburg University, The Netherlands, Tech. Rep., 2009.

[6] R. Coifman and S. Lafon, "Diffusion Maps," *Applied and Computational Harmonic Analysis*, 2006.

[7] J. de la Porte, B. M. Herbst, W. Hereman, and S. J. van der Walt, "An Introduction to Diffusion Maps," in *In The 19th Symposium of the Pattern Recognition Association of South Africa*, 2008.

[8] B. Bah, "Diffusion Maps: Analysis and Applications," Master's thesis, Wolfson College, University of Oxford, 2008.

[9] T. Sipola, "Knowledge Discovery Using Diffusion Maps," Ph.D. dissertation, University of Jyväskylä, 2013.

[10] S. Roweis and L. Saul. An Introduction to Locally Linear Embedding.

[11] M. Belkin and P. Niyogi, "Laplacian Eigenmaps for Dimension Reduction and Data Representation," *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[12] R. Pandit and A. Shehu, "A Principled Comparative Analysis of Dimension Reduction Techniques on Protein Structure Decoy Data," in *International Conference on Bioinformatics and Computational Biology*, Las Vegas, NV, 2016, pp. 43–48.

[13] A. Posadas, F. Vidal, F. de Miguel, G. Alguacil, J. Pena, J. Ibanez, and J. Morales, "Spatial-temporal Analysis of a Seismic Series Using the Principal Components Method," *Journal of Geophysical Research*, pp. 1923–1932, 1993.

[14] M. Turk and A. Pentland, "Face Recognition Using Eigenfaces," *Vision and Pattern Recognition 1991*, pp. 586–591, 1991.

[15] K. Kim, K. Jung, and H. Kim, "Face Recognition Using Kernel Principal Component Analysis," *IEEE Signal Processing Letters*, pp. 40–42, 2002.

[16] G. Sanguinetti, "Dimensionality Reduction of Clustered Datasets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 535–540, 2008.

[17] N. Rajpoot, M. Arif, and A. Bhalerao, "Unsupervised Learning of Shape Manifolds," in *Proceedings of the British Machine Vision Conference*, 2007.

[18] D. Duncan, R. Talmon, H. Zaveri, and R. Coifman, "Identifying Preseizure State in Intracranial EEG Data Using Diffusion Kernels," *Math Biosci Eng.*, pp. 579–590, June 2013.

[19] H. Chang, D. Yeung, and Y. Xiong, "Super-Resolution Through Neighbor Embedding," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2004, pp. 275–282.

[20] R. Duraiswami and V. Raykar, "The Manifolds of Spatial Hearing," in *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, vol. 3, 2005, pp. 285–288.

[21] X. He, D. Cai, S. Yan, and H.-J. Zang, "Neighborhood Preserving Embedding," in *Proceedings of the 10th IEEE International Conference on Computer Vision*, 2005, pp. 1208–1213.

[22] *Coloring of DT-MRI Fiber Traces Using Laplacian Eigenmaps*, 2003.

[23] W.-L. Chao, "Dimensionality Reduction," Graduate Institute of Communication Engineering, National Taiwan University, Tech. Rep., 2011. [Online]. Available: http://disp.ee.ntu.edu.tw/~pujols/Dimensionality%20Reduction.pdf